Using Software Structure to Predict Vulnerability Exploitation Potential

[1]Awad A. Younis and [1]Yashwant K. Malaiya
[1]Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA
{younis,malaiya}@cs.colostate.edu

***Abstract*— Most of the attacks on computer systems are due to the presence of vulnerabilities in software. Recent trends show that number of newly discovered vulnerabilities still continue to be significant. Studies have also shown that the time gap between the vulnerability public disclosure and the release of an automated exploit is getting smaller. Therefore, assessing vulnerabilities exploitability risk is critical as it aids decision-makers prioritize among vulnerabilities, allocate resources, and choose between alternatives. Several methods have recently been proposed in the literature to deal with this challenge. However, these methods are either subjective, requires human involvement in assessing exploitability, or do not scale. In this research, our aim is to first identify vulnerability exploitation risk problem. Then, we introduce a novel vulnerability exploitability metric based on software structure properties viz.: attack entry points, vulnerability location, presence of dangerous system calls, and reachability. Based on our preliminary results, reachability and the presence of dangerous system calls appear to be a good indicator of exploitability. Next, we propose using the suggested metric as feature to construct a model using machine learning techniques for automatically predicting the risk of vulnerability exploitation. To build a vulnerability exploitation model, we propose using Support Vector Machines (SVMs). Once the predictor is built, given unseen vulnerable function and their exploitability features the model can predict whether the given function is exploitable or not.**

***Keywords*—**Risk Assessment; Measurement; Software Vulnerability; Software Security Metrics; Attack Surface, Machine Learning.

1. Introduction

Recent trends show that the number of newly discovered vulnerabilities still continue to be significant (+13000 vulnerabilities in 2013) and so does the number of attacks (+37 million of attacks in 2012-2013) [1]–[3]. It has also been observed that the time gap between the vulnerability public disclosure and the release of an automated exploit is getting smaller [4]. Therefore, assessing the risk of exploitation associated with software vulnerabilities is needed to aid decision-makers prioritize among vulnerabilities, allocate resources, and choose between alternatives.

Several methods have been proposed to deal with this challenge. They can be categorized into: test-based, measurement-based, model-based, and analysis-based approaches. First, test-based approaches do not scale up as writing an exploit is difficult and expensive. Second, measurement-based approaches either make the implicit assumption that all vulnerabilities have the same risk of exploitation (number of attack entry points), or are subjective in nature and do not model properties of software structure. Third, model-based approaches also assume that vulnerability have the same risk of exploitation (number of vulnerabilities).This is unrealistic because different vulnerabilities have different chances of being exploited depending upon their inherent properties such as reachability. Besides, model-based approaches do not model properties of software structure. Finally, analysis-based approaches either require human involvement in the assessment or require vulnerability intelligence provider.

In this research proposal, our aim is to reduce subjectivity, minimize human involvement and improve scalability in assessing vulnerability exploitation risk. To reduce subjectivity, we introduce a novel vulnerability exploitability metric based on software structure properties such as attack entry points, vulnerability location, presence of dangerous system calls, and reachability for less subjective measures. Based on the preliminary results, reachability and the presence of dangerous system calls appear to be good indicator of exploitability.

To reduce human involvement and improve scalability in assessing exploitability risk, we propose using the suggested metric as feature to construct a model using machine learning techniques. We plan to examine the effectiveness of machine learning for automatically predicting the risk of vulnerability exploitation. To build a vulnerability exploitation model, we consider using Support Vector Machines (SVMs). Once the predictor is built, given a vulnerable function and their exploitability features the predictor can assess whether it is exploitable or not and estimate the impact of its exploitation.

The paper is organized as follows. Section 2 presents the problem description and research motivations. In section 3, related works is discussed. In Section 4, the proposed metric will be discussed. In the following section, the key steps of our framework are introduced. Section 6 presents the preliminary results. Finally, the concluding comments are given along with the issues that need further research.

2. Problem Description and Research Motivation

Vulnerability exploitation risk depends on the likelihood of exploiting a vulnerability and the effect of this exploitation, as given in Eq.1. The first factor, exploitability, is the likelihood that a potential vulnerability can be successfully exploited. This factor concerns to the question *"Is the vulnerability exploitable?"* and thus is a classification problem. The other factor, impact, means the losses that occur given a successful exploitation. This factor is related to the question *"Which is the most exploitable vulnerability?*" and hence is a ranking problem.

$$\boldsymbol{Exploitation\ Risk} = \boldsymbol{f}(\boldsymbol{Exploitability}, \boldsymbol{Impact}) \quad (\mathbf{1})$$

To clarify why this problem is challenging, we will look at the main issues that contribute to making the problem complicated. We will use [5], [6] as guide for our analysis of the problem. Here, the problem will be expressed as questions. First we will look at the challenges of the exploitability factor and identify seven key questions that are needed to be addressed. Then, we will discuss the challenges of impact factor and also identify one key question that is required to be tackled.

2.1 Exploitability Factor

There are four main challenges when it comes to assessing the exploitability factor. They are explained as follows.

1) Exploitability Estimators

The main challenge is determining the estimator (attribute) to be used in assessing exploitability factor. There are number of estimators such as: existence of exploit, existence of patches, number of vulnerabilities, number of attack entry points, black market, and Proof of concept.

978-1-4799-5843-6/14 $31.00 © 2014 IEEE
DOI 10.1109/SERE-C.2014.17

Table 1: Exploitability Estimators Limitations.	
Exploitability Estimator	**Limitation**
a. Existence of an exploit	The data required to measure this attribute is not always available.
b. Existence of a patch	Existence of a patch does not tell whether they have been applied or not because studies [7] shows patches applications depends on the administrators' behavior.
c. Number of vulnerabilities and number of attack entry points	These estimators are not informative as they do not specify which vulnerability is exploitable and rather they estimate the exploitability of the whole system.
d. Black market	It is expensive as it requires continues intelligence gathering. Besides, attackers do not tend to share their expertise with the public and their activities are most of the time unknown.
e. Proof of concept	It is hard, not scalable, and expensive to generate reliable exploit.

Q1: Which one of those makes a good estimator of exploitability?

To answer this question, let's look at every one of those estimators. As it can be seen from Table 1, every one of the estimators assesses exploitability from specific prospective and has its own limitation. As no single estimator can capture the whole picture of exploitability, incorporating multiple estimators to form a complete understanding of the exploitation risk is desired. However, the challenge then:

Q2: How can we combine those estimators?

2) Measures derivation

The other challenging is deriving the measures of vulnerability exploitation. Obtaining the measures of exploitability can be accomplished by using one of the following: security expert opinion, history of reported vulnerabilities and exploits, and software code. In one hand, relying on expert opinions leads to subjectivity and thus hinders the accuracy of the assessment. The challenge is:

Q3: How can we reduce subjectivity and minimize human involvement in exploitation assessment?

On the other hand, the history data of reported vulnerabilities or exploits is not always available especially for newly released software. Thus, the question is:

Q4: How can risk of exploitation get assessed in the absence of history data?

The alternative could be using the software code. However, the question is:

Q5: What type of features can be used as an indicative of vulnerability exploitability?

3) Assessment Method

Another challenge is choosing the assessment method. The chosen method should capture the essence of the problem and its related aspects. The challenge is:

Q6: Which one of the following methods: testing, modeling, measurement, and analysis should be used?

Answering this question requires looking at what the requirements of the problem are. Based on the above mentioned challenges, the method should satisfy the following criteria.

- The method should be able to combine multiple estimators and also should able to be customized once a new estimator is introduced.
- The method should reduce subjectivity.
- The method should minimize human involvement in performing assessment (Automation).
- The method should rely on multiple sources of data.
- The method should be speedy and systematic in performing the assessment.

Table 2 compares the assessment methods based on the criteria stated above. This can be used as a guide for choosing the right method.

4) Level of assessment.

A further challenge is deciding on the ***level*** of assessment that the exportability should be assessed at.

Q7: Should we assess exploitability for individual vulnerabilities or the whole software?

Assessing the vulnerability exploitability at the software level is not informative as it assumes that all vulnerability have the same risk of exploitation. This is unrealistic as different vulnerabilities have different risk of exploitation. Thus, assessing the exploitation risk at the individual vulnerability level should be done first. Thereafter, we can add up the number of the exploitable vulnerabilities and hence get the total risk of exploitation for the whole system.

Table 2: Comparison of the Assessment Methods.				
Method / Criteria	**Testing**	**Measurement**	**Modeling**	**Analysis**
Adapt to adding new features	No	No	Yes	No
Reduce subjectivity	Yes	Yes	Yes	No
Minimize human involvement	Yes	No	Yes	No
Use multiple sources of data	No	No	Yes	Yes
Speedy	Yes	No	Yes	No

2.2 Impact Factor

Estimating the impact factor is challenging because it is a context dependent. For instance, a mission critical server being shut down may be more "severe" than a print server. There are two types of impacts: Technical impact (e.g., privilege elevation) and Business impact (e.g., monetary loss). While the latter depends on the mission and the priority of the given context, the former, however, can be estimated at function level. Nevertheless, the question is:

Q8: What estimators should be used to determine the technical impact?

The answer could be any one of those: privilege, access right, dangerous system calls, and exploit mitigation. While determining the dangerous system calls can be easily accomplished at the source code level, privilege and access right is hard as they require deep analysis of the source code. Besides, in the absence of the source code the problem gets even harder because identifying the

privilege and access right from a binary is not a trivial task. However, identifying the exploit mitigation at the function level is possible when the source code is available but it is hard when all you have is an executable file.

3. RESEARCH OBJECTIVES

The main objective of this research is to propose a framework that can encounter for the challenging questions discussed in the problem description. We mainly focus on reducing subjectivity, minimize human involvement and improve scalability in assessing vulnerability exploitation risk. To reduce subjectivity, we introduce novel vulnerability exploitability metric based on software properties: attack entry points, vulnerability location, dangerous system calls, and reachability analysis. To minimize the human involvement and improve scalability, we construct a model based on machine learning techniques that uses the proposed metric as feature to predict the risk of vulnerability exploitation.

4. RELATED WORK

In this section, we review the work related to vulnerability exploitation risk assessment. We organize this section based on the method used to assess exploitation risk into: measurement-based, model-based, test-based, and analysis-based approaches.

4.1 Measurement-based approaches

Attack Surface Metric: The attack surface notion was first introduced by Howard in his Relative Attack Surface Quotient metric [8]. It was later formally defined by Manadhdata and Wing in [9]. They proposed a framework that included the notion of Entry and Exit Points and the associated damage potential-effort ratio. They have applied their formally defined metric to many systems and the results show the applicability of the notion of attack surface. Their new metric has been adapted by a few major software companies, such as Microsoft, Hewlett-Packard, and SAP. Manadhdata et al in [10] relate the number of reported vulnerabilities for two FTP daemons with the attack surface metric along the method dimension. Younis and Malaiya [11] have compared vulnerability density of two versions of Apache HTTP server with the attack surface metric along the method dimension. However, attack surface metric does not measure the risk of exploitation for individual vulnerabilities. Rather, it measures the exploitability for the whole system and as a result it cannot help in prioritizing among vulnerabilities. Besides, neither [10] nor [11], however, related entry points with the location of the vulnerability to measure its exploitability.

CVSS Metrics: CVSS metrics are the de facto standard that is currently used to measure the severity of vulnerabilities [12]. CVSS Base Score measures severity based on exploitability (the ease of exploiting vulnerability) and impact (the effect of exploitation). Exploitability is assessed based on three metrics: Access Vector, Authentication, and Access Complexity. However, CVSS exploitability measures have come under some criticism. First, they assign static subjective numbers to the metrics based on expert knowledge regardless of the type of vulnerability, and they do not correlate with the existence of known exploit [13]. Second, two of its factors (Access Vector and Authentication) have the same value for almost all vulnerabilities [14]. Third, there is no formal procedure for evaluating the third factor (Access Complexity) [12]. Consequently, it is unclear if CVSS considers the software structure and properties as a factor.

4.2 Model-based approaches

Probabilistic Model: Joh and Malaiya in [15] formally defined a risk measure as a likelihood of adverse event and the impact of this event. In one hand, they utilized the vulnerability lifecycle and applied Markov stochastic model to measure the likelihood of vulnerability exploitability for an individual vulnerability and the whole system. On the other hand, they used the impact related metrics from CVSS to estimate the exploitability impact. They applied their metric to assess the risk of two systems that had known unpatched vulnerabilities using actual data. However, the transition rate between vulnerability lifecycle events has not been determined and the probability distribution of lifecycle events remains to be studied. Moreover, the probability of being in an exploit state requires information about the attacker behavior which might not be available. Additionally, the probability of a patch being available but not applied requires information about the administrator behavior which has not been considered by the proposed model and also hard to be obtained. In contrast, we assess vulnerability exploitability for individual vulnerabilities based on code properties regardless of the availability or unavailability of a patch.

Logistic Model: Vulnerability density metric assesses the risk of potential exploitation based on the density of the residual vulnerabilities [16]. The density of residual vulnerabilities is measured based on the number of known reported vulnerabilities and the total number of vulnerabilities. However, the total number of vulnerabilities is unknown but can be predicted using vulnerability discovery models (VDMs). Alhazmi and Malaiya [17] proposed a logistic vulnerability discovery model, termed the AML model. AML examines the reported known vulnerabilities of a software system to estimate the total number of vulnerabilities and their rate of discovery. However, considering the number of vulnerabilities alone is insufficient in assessing the risk of individual vulnerability exploitation. Because different vulnerabilities have different opportunity of being exploited based on their properties such as reachability.

Machine Learning based Metric: Bozorgi et al. [13] aimed at measuring vulnerabilities severity based on likelihood of exploitability. They argued that the exploitability measures in CVSS Base Score metric cannot tell much about the vulnerability severity. They attributed that to the fact that CVSS metrics rely on expert knowledge and static formula. To that end, the authors proposed a Machine Learning and Data mining technique that can predict the possibility of vulnerability exploitability. They observed that much vulnerability have been found to have high severity score using CVSS exploitability metric although there were no known exploits existing for them. This indicates that CVSS score does not differentiate between exploited and non-exploited vulnerabilities. This result has also been confirmed by Allodi et al. [14], [18], [19]. However, unlike their work, ours relies on software properties such as attack surface entry points, source code structure, and the vulnerabilities location to estimate vulnerability exploitability. This is particularly important for newly released applications that do not have large amount of historical vulnerabilities.

4.3 Test-based approaches (Proof of concept)

Automated exploit-generation system (AEG): T. Avgerinos et al. [20] proposed an automated exploit-generation system (AEG) to assess the risk of vulnerability exploitation. AEG first uses static analysis to find potential bug locations in a program, and then uses a combination of static and dynamic analysis to find an execution path that reproduces the bug, and then generates an exploit automatically. AEG generates exploits, which provide evidence that the bugs it finds are critical security vulnerabilities. However, generating an exploit is expensive and does not scale. AEG has only been applied to specific type of vulnerabilities and software.

Black Fuzz Testing: Sparks et al in [21] extended the black box fuzzing using a genetic algorithm that use the past branch profiling information to direct the input generation in order to cover specified program regions or points in the control flow graph. The control flow is modeled as Markov process and fitness function is defined over Markov probabilities which are associated with state transition on control flow graph. They generated inputs using grammatical evolution. These inputs are capable of reaching deeply vulnerable code which is hidden in a hard to reach locations. In contrast to their work, ours relies on source code analysis, a link between vulnerability location and attack surface entry points, and dangerous system call analysis that were specifically intended for measuring vulnerability exploitability.

4.4 Analysis-Based approaches

Black Market Data Analysis: L. Allodi and F. Massacci in [14], [18] proposed the black market as an index of risk of vulnerability exploitation. Their approach assesses the risk of vulnerability exploitation based on the volumes of the attacks coming from the vulnerability in the black market. It first looks at the attack tools and verifies whether the vulnerability is used by such tool or not. It also analyzes the attacks on the wild to verify whether the vulnerability have been a target of such attacks or not. If the vulnerability is being used by one of the attack tools or being a target of real attacks, they consider this vulnerability as a threat for exploitation. This approach has introduced a new view of measuring risk of exploitation by considering the history of attacks at vulnerabilities. This approach does not require spending large amount of technical resources to thoroughly investigate the possibility of vulnerability exploitation. However, this approach requires vulnerability intelligence provider as the information about the attacks and tools are dynamic in nature. Moreover, if the vulnerability right now is not used by a tool or it is not a target of an attack, it does not mean that it is going to be so continually. Our approach, on the other hand, relies only on software properties and does not make any assumption about the attacks and attacker resources.

Source Code Analysis: Brenneman [22] has introduced the idea of linking the attack surface entry point to the attack target to prioritize the effort and resource required for software security analysis. Their approach is based on path-based analysis, which can be utilized to generate an attack map. This helps visualizing the attack surfaces, attack target, and functions that link them. This is believed to make significant improvement to software security analysis. In contrast to their work, we not only utilize the idea of linking attack surface entry point with the reported vulnerability location to estimate vulnerability exploitability, but also apply the damage potential-effort ratio in the attack surface metric and checked for the dangerous system calls inside every related entry point to estimate how likely the entry point is going to be used in an attack. This is helpful for inferring attacker's motive in invoking the entry point method.

System Calls Analysis: E.Gabrielli and L.Mancini in [23] have presented a detailed analysis of the UNIX system calls and classify them according to their level of threat with respect to system penetration. To control these system calls invocation, they proposed Reference Monitor for UNIX System (REMUS) mechanism to detect intrusion that may use these system calls which could subvert the execution of privileged applications. Nevertheless, our work applies their idea to deduce the motive of an attacker in using an entry point, as attackers usually looks to cause more damage to targeted systems. Thus, our work is not about intrusion detection but rather measuring the exploitability of a known vulnerability.

5. PROPOSED METRIC

Security is defined as "the freedom from the possibility of suffering damage or loss from malicious attack [24]." Quantitative security is realized by means of measurement. A measurement of a security is represented by a metric. A metric is a system of related measures enabling quantification of some characteristics [24]. A security metric is a quantifiable measurement that indicates the level of security. *We are interested in studying security metrics and in particular metrics for vulnerability exploitation prediction.*

Our metric is based on combining attack surface analysis, vulnerability analysis, and exploitation analysis to assess exploitability. The proposed metric uses three values: high, medium, and low as a measure of exploitability risk. The values are assigned based on the following. *High*, if a vulnerability is reachable from an entry point with dangerous system calls. *Medium*, if it is reachable from an entry point with no dangerous system calls. *Low*, if it is not reachable from any entry points. The following shows the steps used to obtain the measures of our metric. Further details can be found in [25].

5.1 Define attack entry points of software

We define the attack entry points using the system's attack surface entry point framework in is [9]. Entry points are the functions that an attacker uses to send data to the system. In this paper, we used only the entry points as they are the main target by malicious attacks. A function is a direct entry point if it receives data directly from the environment; read method defined in unistd.h in C library is an example [26]. Fig.1 shows how the entry points are identified. Input functions are function that receives data from the environment.

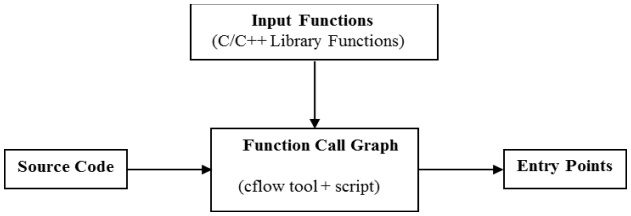

Figure 1: Attack Entry Points Identification

The required steps are explained as follows. After we obtain the source code, we first identify all functions that receive data from the user environment (C/C++ Library functions). Then, we verify whether these functions are used by any of the user functions. After that, we identify all functions called by the main function using cflow. By using python script, we verify whether any of these functions has one of the C/C++ input functions. If we find any, we consider that function as EP. Finally, we get the list of all identified entry points

5.2 Finding Vulnerability Location

The vulnerability location can be found by manually looking at the report or automatically by using static code analyzer such as Splint [27]. The flowing report shows how a location can be found from a vulnerability report.

CVE-2011-0419:

"Stack consumption vulnerability in the fnmatch implementation in ap_fnmatch.c in the Apache Portable Runtime (APR) library before 1.4.3 and the Apache HTTP Server before 2.2.18."

5.3 Reachability Analysis

We employed a *system dependence graph* (SDG) in [28] to determine the calls from an entry point function to a vulnerability location (vulnerable function). SDG represents programs in a graph that includes functions and function calls. We have used cflow tool [29] to generate this graph. Fig.2 shows reachability analysis for one of the chosen vulnerabilities.

5.4 Dangerous System Calls

We estimate how likely an entry point is going to be used in an attack using Dangerous System Calls (DSCs) proposed in [23]. DSCs are specific system calls that have been identified and classified into four levels [23]. Level one allows full control of the system while level two used for denial of service attack. On the other hand, level three used for disrupting the invoking process and level four is considered harmless. The following system calls are an example of threat level one: *mount, open, and link*. A complete list of these calls can be found in [23]. However, from the list of the identified entry points, we verify whether the entry point contains DSCs or not using a python script. If any DSCs are found, we annotate that function as entry point with DSCs.

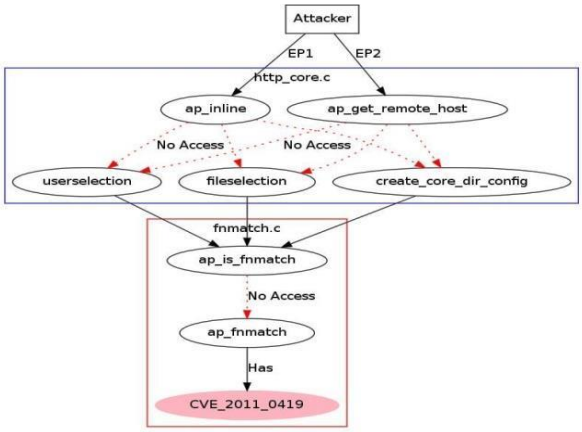

Figure 2: Directly Mapping the EP in http_core.c to the vulnerable method.

6. Proposed Framework

Fig.3 shows the framework of our proposed method. This framework is based on three layers as they will be explained next.

6.1 Feature Extraction (Layer 1)

First, we start by mapping vulnerabilities to their function (location). This is accomplished by using the vulnerability and bug repository databases. When the vulnerability location is not available, we can use static code analyzer like (e.g. Splint) to map vulnerabilities to their locations (functions). Next, we extract a set of features from these functions. These features are extracted as shown in section 5. Our proposed metric evaluates vulnerability exploitability based on the presence of a function call connecting attack surface entry points to the vulnerability location within the software under consideration using SDG. Further details about the metric can be found in [25]. If such a call exists, we estimate whether the entry point is going to be used in an attack based on dangerous system calls [30]. The dangerous system calls paradigm has been considered as these system calls allow attackers to escalate a method privilege and hence cause more damage.

6.2 Model Building (Layer 2)

To build a vulnerability exploitation predictor based on the selected features, we model vulnerability exploitation prediction as a supervised learning problem. Supervised learning is one of machine learning methods in which a set of labeled examples is used to learn a target function. The target function maps the inputs to a desired set of outputs (labels). Input to a supervised classification algorithm is a set of training data $S = \{s_1, s_2, \ldots, s_n\}$. Each vector $s_i = \{x_i^1, x_i^2, \ldots, x_i^m\}$, $c_i \in S$ is called a training instance, where x_m is a feature and c_i is the class label of the training instance s_i. We propose using Support Vector Machines (SVMs) as a classifier. SVM [31] is a supervised learning algorithm that is used for classification. SVM creates a hyper-plane in a high dimensional space that can separate the instances in the training set according to their class labels. When a linear separation cannot be found in the original feature space, SVMs use kernel functions to map training data into a higher dimensional feature space. Then SVM creates a linear separator in this higher dimensional feature space, which can be used to classify unseen data instances.

Table 3 represents an example of data set that will be used for learning the classifier. The feature value of $feat_i$ represents reachability and dangerous system calls features extracted using our proposed metric. The feature value of $feat_n$ for the function f_n represented by v_{nm}. The c_n represents the class label for the function f_n indicting whether a function is exploitable or not (1 or -1).

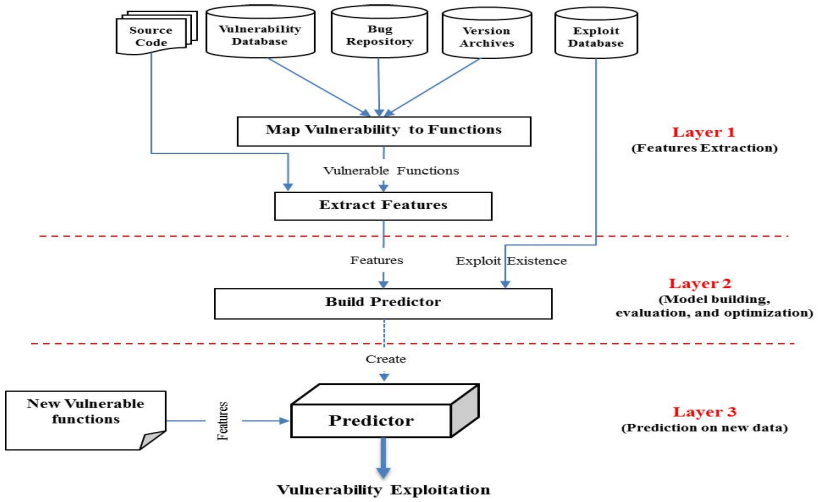

Figure 3: Framework to predict vulnerability exploitation from reachability Metric

Table4: The Dataset of the Proposed Method.				
Vulnerability	**Vulnerable Function**	**Reachability**	**Dangerous System Calls**	**Exploit Existence**
1. CVE-2012-0031	ap_cleanup_scoreboard	Reachable	setuid, open, fork, kill, exit, unlink, setgid, dup, and flock	No Exploit
2. CVE-2010-0010	ap_proxy_send_fb	Reachable	-	Exploit
3. CVE-2004-0488	ssl_util_uuencode_binary	Not reachable	-	No Exploit

6.3 Prediction on new Data (Layer 3)

Finally, the developed predictor is used to predict whether or not unseen vulnerable function will be exploitable or not based on its exploitability features. Using dangerous system calls as a feature implicitly captures the impact factor.

Table 3: Prediction Features.					
Function	$feat_1$	$feat_2$	..	$feat_n$	Class
f_1	v_{11}	v_{12}		v_{1n}	c_1
f_2	v_{21}	v_{22}		v_{2n}	c_2
.	.	.		.	.
f_n	v_{n1}	v_{n2}		v_{nm}	c_n

7. PRELIMINARY RESULTS

7.1 Data Collection

To measure the effectiveness of our proposed method, we built a dataset containing 20 vulnerable functions of Apache HTTP server. We collected the vulnerabilities from National Vulnerability database [2] and the exploits from exploit database [32] and Open Source Vulnerability Database [33]. The source codes of selected software have been obtained from their archive database [30] and [31]. Due to pages limitation, we show only a sample of our dataset in Table 4. The following has been observed from the dataset:

- Two out of the twenty vulnerabilities are not reachable and have no exploits and hence have been assigned low severity values.
- Thirteen out of the twenty vulnerabilities are reachable with no exploit exist for them. More than half of these vulnerabilities have dangerous system calls and hence have been assigned high severity values.
- The remaining five vulnerabilities are reachable with exploits exist for them. Three out of them have dangerous system calls and thus have been assigned high severity values.
- More than half of the vulnerabilities have dangerous system calls.

8. CONCLUSIONS AND FUTURE WORK

In this work, we address the problem of quantifying vulnerability exploitation and identify the limitations of the current state of the art. We proposed a new metric that can be used as an earlier indictor of vulnerability exploitation based on software structure properties. We also proposed utilizing this metric as a feature for building a model. We propose to develop a model that uses machine learning techniques to predict whether a given vulnerability is likely to be exploitable or not. The developed model can help decision makers prioritize their actions objectively based on function structure features. Preliminary results in using the metric as early indicator of exploitability have been shown. We also have discussed how the features can be used to build the classifier.

Our future plans include using our dataset to measure the effectiveness of our model. To judge the performance of the proposed predictive model, we will evaluate the accuracy, the area under the receiver operating characteristics curve (AUC), and false positive rate (FPR) as measures. We will also consider adding more exploitable features at the function level such as Node Rank [36] and GuardStack [37] .

REFERENCES

[1] "HP Report: More Attacks, Despite Fewer New Vulnerabilities Overall - eSecurity Planet." [Online]. Available: http://www.esecurityplanet.com/network-security/hp-report-more-attacks-despite-fewer-new-vulnerabilities-overall.html. [Accessed: 27-Dec-2013].

[2] "National Vulnerability Database," *http://nvd.nist.gov/; July 2013.* .

[3] "Security Response Publications, Internet Security Threat Report | Symantec." [Online]. Available: http://www.symantec.com/security_response/publications/threatreport.jsp. [Accessed: 27-Dec-2013].

[4] S. Frei, B. Tellenbach, and B. Plattner, *0-day patch - exposing vendors (in)security performance," BlackHat Europe, 2008, http://www.techzoom.net/papers/blackhat 0day patch 2008.pdf.* .

[5] G. Stoneburner, A. Goguen, and A. Feringa, "Risk management guide for information technology systems," *Nist special publication*, vol. 800, no. 30, pp. 800–30, 2002.

[6] W. Jansen, *Directions in security metrics research*. NIST, NISTIR 7564, p. 21, 2009.

[7] W. A. Arbaugh, W. L. Fithen, and J. McHugh, "Windows of vulnerability: a case study analysis," *Computer*, vol. 33, no. 12, pp. 52 – 59, Dec. 2000.

[8] M. Howard, J. Pincus, and J. Wing, "Measuring Relative Attack Surfaces," in *Computer Security in the 21st Century*, D. T. Lee, S. P. Shieh, and J. D. Tygar, Eds. Springer US, 2005, pp. 109–137.

[9] P. K. Manadhata and J. M. Wing, "An Attack Surface Metric," *Software Engineering, IEEE Transactions on*, vol. 37, no. 3, pp. 371 –386, Jun. 2011.

[10] P. Manadhata, J. Wing, M. Flynn, and M. McQueen, "Measuring the attack surfaces of two FTP daemons," in *Proceedings of the 2nd ACM workshop on Quality of protection*, New York, NY, USA, 2006, pp. 3–10.

[11] A. A. Younis and Y. K. Malaiya, "Relationship between Attack Surface and Vulnerability Density: A Case Study on Apache HTTP Server," in *in ICOMP, The 2012 International Conference on Internet Computing, 2012.*

[12] P. Mell, K. Scarfone, and S. Romanosky, "A complete guide to the common vulnerability scoring system version 2.0," in *Published by FIRST-Forum of Incident Response and Security Teams*, 2007, pp. 1–23.

[13] M. Bozorgi, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond heuristics: learning to classify vulnerabilities and predict exploits," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2010, pp. 105–114.

[14] L. Allodi and F. Massacci, "My Software has a Vulnerability, should I worry?," *arXiv preprint arXiv:1301.1275*, 2013.

[15] H. Joh and Y. K. Malaiya, "Defining and assessing quantitative security risk measures using vulnerability lifecycle and cvss metrics," in *The 2011 International Conference on Security and Management (sam)*, 2011, pp. 10–16.

[16] O. H. Alhazmi, Y. K. Malaiya, and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems," *Computers & Security*, vol. 26, no. 3, pp. 219–228, 2007.

[17] O. H. Alhazmi and Y. K. Malaiya, "Modeling the vulnerability discovery process," in *Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on*, 2005, p. 10–pp.

[18] L. Allodi and F. Massacci, "A preliminary analysis of vulnerability scores for attacks in wild," *ACM Proc. of CCS BADGERS*, vol. 12, 2012.

[19] L. Allodi, W. Shim, and F. Massacci, "Quantitative assessment of risk reduction with cybercrime black market monitoring," *2013 IEEE Security and Privacy Workshops*, 2013.

[20] T. Avgerinos, S. K. Cha, B. L. T. Hao, and D. Brumley, "AEG: Automatic Exploit Generation.," in *NDSS*, 2011, vol. 11, pp. 59–66.

[21] S. Sparks, S. Embleton, R. Cunningham, and C. Zou, "Automated vulnerability analysis: Leveraging control flow for evolutionary input crafting," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, 2007, pp. 477–486.

[22] D. Brenneman, "Improving Software Security by Identifying and Securing Paths Linking Attack Surface to Attack Target," McCabe Software Inc., White Paper, Apr. 2012.

[23] M. Bernaschi, E. Gabrielli, and L. V. Mancini, "Remus: a security-enhanced operating system," *ACM Trans. Inf. Syst. Secur.*, vol. 5, no. 1, pp. 36–61, Feb. 2002.

[24] O. S. Saydjari, "Is risk a good security metric?," in *Proceedings of the 2nd ACM workshop on Quality of protection*, 2006, pp. 59–60.

[25] A. A. Younis, Y. K. Malaiya, and I. Ray, "Using Attack Surface Entry Points and Reachability Analysis to Assess the Risk of Software Vulnerability Exploitability," presented at the 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering, 2014.

[26] P. Manadhata, J. Wing, M. Flynn, and M. McQueen, "Measuring the attack surfaces of two FTP daemons," in *In Proceedings of the 2nd ACM workshop on Quality of protection, 2006.*

[27] D. Evans and D. Larochelle, "Improving security using extensible lightweight static analysis," *software, IEEE*, vol. 19, no. 1, pp. 42–51, 2002.

[28] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 1, pp. 26–60, 1990.

[29] "GNU cflow." [Online]. Available: http://www.gnu.org/software/cflow/manual/cflow.html. [Accessed: 02-Aug-2013].

[30] M. Bernaschi, E. Gabrielli, and L. V. Mancini, "REMUS: A security-enhanced operating system," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 1, pp. 36–61, 2002.

[31] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[32] "Exploits Database by Offensive Security." [Online]. Available: http://www.exploit-db.com/. [Accessed: 07-Aug-2013].

[33] "OSVDB: Open Sourced Vulnerability Database." [Online]. Available: http://osvdb.org/. [Accessed: 19-Feb-2014].

[34] "archive.apache.org." [Online]. Available: http://archive.apache.org/dist/httpd/. [Accessed: 02-Aug-2013].

[35] "Old Version of Firefox Download - OldApps.com." [Online]. Available: http://www.oldapps.com/firefox.php. [Accessed: 19-Feb-2014].

[36] P. Bhattacharya, M. Iliofotou, I. Neamtiu, and M. Faloutsos, "Graph-based analysis and prediction for software evolution," in *Proceedings of the 2012 International Conference on Software Engineering*, 2012, pp. 419–429.

[37] O. Whitehouse, *Analysis of GS protections in Windows Vista*. Symantec, 2007.